WHAT IS CLAIMED IS:

1. A method for detecting a potential race condition comprising the steps of:

identifying a memory access in a source code segment, the source code segment being executable in any one of a plurality of execution threads;

searching backward from the memory access through each of the plurality of threads for an assert protection declaration without any intervening deassert protection declaration, and identifying the potential race condition if the assert protection declaration without any intervening deassert protection declaration was not found in each of the plurality of threads; and

searching forward from the memory access through each of the plurality of threads for the deassert protection declaration, and identifying the potential race condition if the deassert protection declaration was not found in each of the plurality of threads.

2. The method as recited in claim 1 wherein the step of identifying the memory access further comprises searching the source code segment for a static variable, the static variable being used to identify the memory access.

3. The method as recited in claim 1 wherein the step of identifying the memory access further includes searching the source code segment for a heap variable, the heap variable being used to identify the memory access.

4. The method as recited in claim 1 wherein the step of identifying the memory access further includes prompting a user for the memory access to identify.

5. The method as recited in claim 1 further including the step of identifying a type of protection declaration, and wherein the assert protection declaration is an assert protection declaration of the identified type, and wherein the deassert protection declaration is a deassert protection declaration of the identified type.

6. The method of claim 5, wherein the type of protection declaration is one of an interrupt lock, a task lock, and a semaphore.

7. A method for detecting a potential race condition comprising the steps of:

selecting one or more source code segments, the source code segments being executable in any one of a plurality of execution threads;

selecting a memory access to identify from the selected source code segments;

identifying the memory access in one or more of the selected source code segments;

searching backward from each memory access through each of the plurality of threads for an assert protection declaration without any intervening deassert protection declaration, and identifying the potential race condition if the assert protection declaration without any intervening deassert protection declaration was not found in each of the plurality of threads; and

searching forward from each memory access through each of the plurality of threads for the deassert protection declaration, and identifying the potential race condition if the deassert protection declaration was not found in each of the plurality of threads.

8. The method as recited in claim 7 wherein the step of selecting the memory access further includes searching the source code segments for static variables, the static variables being used to identify the memory access.

9. The method as recited in claim 7 wherein the step of selecting the memory access further includes searching the source code segments for heap variables, the heap variables being used to identify the memory access.

10. The method as recited in claim 7 wherein the step of selecting the memory access further includes prompting a user for the memory access to identify.

11. The method as recited in claim 7 further including the step of identifying a type of protection declaration, and wherein the assert protection declaration is an assert protection declaration of the identified type, and wherein the deassert protection declaration is a deassert protection declaration of the identified type.

12. The method of claim 11, wherein the type of protection declaration is one of an interrupt lock, a task lock, and a semaphore.

13. A method for detecting a potential race condition comprising the steps of:

(a) identifying a memory access in a first source code segment of a plurality of source code segments, the first code segment being a node interconnecting a first tree and a second tree, and each source code segment, except the first source code segment, arranged either in the first tree as an ancestor of the first code segment or in the second tree as a descendant of the first code segment;

(b) selecting the first source code segment;

(c) (1) searching upward through the selected source code segment of the first tree for an assert protection declaration without any intervening deassert protection declaration, and identifying the potential race condition if: i) an assert protection declaration without any intervening deassert protection declaration was not found in the selected source code segment of the first tree and ii) the selected source code segment has no parents;

(c)(2) if the selected source code segment has parents, selecting each parent source code segment of the selected source code segment, and repeating step (c) for each of the selected parent source code segments;

(d) selecting the first source code segment; and

(e)(1)searching downward through the selected source code segment of the second tree for the deassert protection declaration, and identifying the potential race condition if : i) the deassert protection declaration was not found in the selected source code segment of the second tree, and ii) the selected source code segment has no children,

(e)(2) if the selected source code segment has children, selecting each child source code segment of the selected source code segment, and repeating steps (e) for each of the selected parent source code segments.

14. The method as recited in claim 13 wherein the step of identifying the memory access further comprises searching the source code segment for a static variable, the static variable being used to identify the memory access.

15. The method as recited in claim 13 wherein the step of identifying the memory access further includes searching the source code segment for a heap variable, the heap variable being used to identify the memory access.

16. The method as recited in claim 13 wherein the step of identifying the memory access further includes prompting a user for the memory access to identify.

17. The method as recited in claim 13 further including the step of identifying a type of protection declaration, and wherein the assert protection declaration is an assert protection declaration of the identified type, and wherein the deassert protection declaration is a deassert protection declaration of the identified type.

18. The method of claim 17, wherein the type of protection declaration is one of an interrupt lock, a task lock, and a semaphore.

19. The method of claim 17, wherein the type of protection declaration is one of an interrupt lock, a task lock, a mutex, and a semaphore.

20. The method as recited in claim 13 wherein the assert protection declaration is an interrupt lock and the deassert protection declaration is an interrupt unlock.

21. The method as recited in claim 13 wherein the assert protection declaration is a task lock and the deassert protection declaration is an task unlock.

22. The method as recited in claim 13 wherein the assert protection declaration is a take semaphore and the deassert protection declaration is a give semaphore.

23. The method of claim 5, wherein the type of protection declaration is one of an interrupt lock, a task lock, a mutex, and a semaphore.

24. The method as recited in claim 1 wherein the assert protection declaration is an interrupt lock and the deassert protection declaration is an interrupt unlock.

25. The method as recited in claim 1 wherein the assert protection declaration is a task lock and the deassert protection declaration is an task unlock.

26. The method as recited in claim 1 wherein the assert protection declaration is a take semaphore and the deassert protection declaration is a give semaphore.

27. The method of claim 11, wherein the type of protection declaration is one of an interrupt lock, a task lock, a mutex, and a semaphore.

28. The method as recited in claim 7 wherein the assert protection declaration is an interrupt lock and the deassert protection declaration is an interrupt unlock.

29. The method as recited in claim 7 wherein the assert protection declaration is a task

lock and the deassert protection declaration is an task unlock.

30. The method as recited in claim 7 wherein the assert protection declaration is a take semaphore and the deassert protection declaration is a give semaphore.

31. The method as recited in claim 1, wherein the searching steps further include the step of stopping the search when a maximum depth has been reached in the plurality of threads.

32. The method as recited in claim 7, wherein the searching steps further include the step of stopping the search when a maximum depth has been reached in the plurality of threads.

33. The method as recited in claim 13, wherein steps (c) and (e) each include the step of initializing a respective depth counter, and wherein steps (c) and (e) each include the step of incrementing the depth counter, comparing the depth counter to a maximum depth, and stopping the searching if the depth counter is equal to the maximum depth.

34. The method as recited in claim 13, wherein steps (c) and (e) each include the step of initializing a respective depth counter, and wherein steps (c) and (e) each include the step of decrementing the depth counter, comparing the depth counter to a minimum depth, and stopping the searching if the depth counter is equal to the minimum depth.

35. The method as recited in claim 1 wherein the steps of searching are performed recursively.

36. The method as recited in claim 13 wherein the steps of searching are performed recursively.

37. The method as recited in claim 13 wherein steps (c) and (e) are performed

recursively.

38. The method as recited in claim 1 wherein the step of searching backward is conducted in parallel on a plurality of processing devices.

39. The method as recited in claim 1 wherein the step of searching forward is conducted in parallel on a plurality of processing devices.

40. The method as recited in claim 1 wherein the steps of searching backward and forward are conducted in parallel on a plurality of processing devices.

41. The method as recited in claim 13 wherein step (c) is conducted in parallel on a plurality of processing devices.

42. The method as recited in claim 13 wherein step (e) is conducted in parallel on a plurality of processing devices.

43.The method as recited in claim 13 wherein steps (c) and (e) are conducted in parallel.

44.The method as recited in claim 7 wherein the step of searching backward is conducted in parallel on a plurality of processing devices.

45.The method as recited in claim 7 wherein the step of searching forward is conducted in parallel on a plurality of processing devices.

46.The method as recited in claim 7 wherein the steps of searching backward and forward are conducted in parallel.

47.A computer readable medium, having stored thereon, computer executable process

steps operative to control a computer to detect possible race conditions, the process steps comprising:

identifying a memory access in a source code segment, the source code segment being executable in any one of a plurality of execution threads;

searching backward from the memory access through each of the plurality of threads for an assert protection declaration without any intervening deassert protection declaration, and identifying the potential race condition if the assert protection declaration without any intervening deassert protection declaration was not found in each of the plurality of threads; and

searching forward from the memory access through each of the plurality of threads for the deassert protection declaration, and identifying the potential race condition if the deassert protection declaration was not found in each of the plurality of threads.

48.A computer readable medium, having stored thereon, computer executable process steps operative to control a computer to detect possible race conditions, the process steps comprising:

selecting one or more source code segments, the source code segments being executable in any one of a plurality of execution threads;

selecting a memory access to identify from the selected source code segments;

identifying the memory access in one or more of the selected source code segments;

searching backward from each memory access through each of the plurality of threads for an assert protection declaration without any intervening deassert protection declaration, and identifying the potential race condition if an assert protection declaration without any intervening deassert protection declaration was not found in each of the plurality of threads; and

searching forward from each memory access through each of the plurality of threads for the deassert protection declaration, and identifying the potential race condition if the deassert protection declaration was not found in each of the plurality of

threads.

49.A computer readable medium, having stored thereon, computer executable process steps operative to control a computer to detect possible race conditions, the process steps comprising:

(a) identifying a memory access in a first source code segment of a plurality of source code segments, the first code segment being a node interconnecting a first tree and a second tree, and each source code segment, except the first source code segment, arranged either in the first tree as an ancestor of the first code segment or in the second tree as a descendant of the first code segment;

(b) selecting the first source code segment;

(c) searching upward through the selected source code segment of the first tree for an assert protection declaration without any intervening deassert protection declaration, and identifying the potential race condition if: i) an assert protection declaration without any intervening deassert protection declaration was not found in the selected source code segment of the first tree and ii) the selected source code segment has no parents;

(c)(1) if the selected source code segment has parents, selecting each parent source code segment of the selected source code segment, and

(c)(1)(A) repeating step c for each of the selected parent source code segments;

(d) selecting the first source code segment; and

(e) searching downward through the selected source code segment of the second tree for the deassert protection declaration, and identifying the potential race condition if : i)

the deassert protection declaration was not found in the selected source code segment of the second tree, and ii) the selected source code segment has no children,

(e)(1) if the selected source code segment has children, selecting each child source code segment of the selected source code segment, and

(e)(1)(A) repeating steps e for each of the selected parent source code segments.

50. A system comprising:

a processing device coupled to a memory, the processing device operable to:

identify a memory access in a source code segment, the source code segment being executable in any one of a plurality of execution threads;

search backward from the memory access through each of the plurality of threads for an assert protection declaration without any intervening deassert protection declaration, and identify the potential race condition if the assert protection declaration without any intervening deassert protection declaration was not found in each of the plurality of threads; and

search forward from the memory access through each of the plurality of threads for the deassert protection declaration, and identify the potential race condition if the deassert protection declaration was not found in each of the plurality of threads.

51. A system comprising:

a processing device coupled to a memory, the processing device operative to:

select one or more source code segments, the source code segments being executable in any one of a plurality of execution threads;

select a memory access to identify from the selected source code segments;

identify the memory access in one or more of the selected source code segments;

search backward from each memory access through each of the plurality of

threads for an assert protection declaration without any intervening deassert protection declaration, and identify the potential race condition if an assert protection declaration without any intervening deassert protection declaration was not found in each of the plurality of threads; and

search forward from each memory access through each of the plurality of threads for the deassert protection declaration, and identify the potential race condition if the deassert protection declaration was not found in each of the plurality of threads.

52. A system comprising:

a processing device coupled to a memory, the processing device operative to:

(a) identify a memory access in a first source code segment of a plurality of source code segments, the first code segment being a node interconnecting a first tree and a second tree, and each source code segment, except the first source code segment, arranged either in the first tree as an ancestor of the first code segment or in the second tree as a descendant of the first code segment;

(b) select the first source code segment;

(c) search upward through the selected source code segment of the first tree for an assert protection declaration without any intervening deassert protection declaration, and identify the potential race condition if: i) an assert protection declaration without any intervening deassert protection declaration was not found in the selected source code segment of the first tree and ii) the selected source code segment has no parents;

(c)(1) if the selected source code segment has parents, select each parent source code segment of the selected source code segment, and

(c)(1)(A) repeat step c for each of the selected parent source code segments;

(d) select the first source code segment; and

(e) search downward through the selected source code segment of the second tree for the deassert protection declaration, and identify the potential race condition if : i) the deassert protection declaration was not found in the selected source code segment of the second tree, and ii) the selected source code segment has no children,

(e)(1) if the selected source code segment has children, select each child source code segment of the selected source code segment, and

(e)(1)(A) repeat steps e for each of the selected parent source code segments.

53. The system as recited in claim 50 wherein the processing device comprises a plurality of processors operating in parallel.

54. The system as recited in claim 51 wherein the processing device comprises a plurality of processors operating in parallel.

55. The system as recited in claim 52 wherein the processing device comprises a plurality of processors operating in parallel.

56. The system as recited in claim 50 wherein the memory comprises a plurality of interconnected memory devices.

57. The system as recited in claim 51 wherein the memory comprises a plurality of interconnected memory devices.

58. The system as recited in claim 52 wherein the memory further comprises a plurality of interconnected memory devices.

59. The system as recited in claim 53 wherein the memory further comprises a plurality of interconnected memory devices.

60. The system as recited in claim 54 wherein the memory further comprises a plurality of interconnected memory devices.

61. The system as recited in claim 55 wherein the memory further comprises a plurality of interconnected memory devices.

62. The system as recited in claim 51, wherein the processing device includes a plurality of processing devices and wherein at least two of the plurality of threads are searched backward in parallel on different ones of the plurality of processing devices.

63. The system as recited in claim 51 wherein the processing device includes a plurality of processing devices and wherein at least two of the plurality of threads are searched forward in parallel on different ones of the plurality of processing devices.

64. The system as recited in claim 52, wherein the processing device includes a plurality of processing devices and wherein at least two of the plurality of threads are searched backward in parallel on different ones of the plurality of processing devices.

65. The system as recited in claim 52 wherein the processing device includes a plurality of processing devices and wherein at least two of the plurality of threads are searched forward in parallel on different ones of the plurality of processing devices.

66. The system as recited in claim 53 wherein the processing device includes a plurality of available processing devices, and wherein each selected source code segment of (c)(1) up to a number of selected source code segments equal to the plurality of available processing devices is searched upward on a different one of the plurality of

available processing devices.

67. The system as recited in claim 53 wherein the processing device includes a plurality of available processing devices, and wherein each selected source code segment of (e)(1) up to a number of selected source code segments equal to the plurality of available processing devices is searched downward on a different one of the plurality of available processing devices.

68.The system as recited in claim 13 wherein the processing device includes a plurality of available processing devices, and wherein each selected source code segment of (e)(1) and (c)(1) up to a number of selected source code segments equal to the plurality of available processing devices is searched on a different one of the plurality of available processing devices.

69. A system comprising:

a race condition detection module configured to detect a memory access for a variable in a source file;

an assertion determination module configured to receive from the race condition detection module the source file and configured to identify whether assertion of protection has been provided in all the threads of execution that use the memory access; and

a deassertion determination module configured to receive from the race condition determination module the source file and configured to identify whether deassertion of protection has been provided in all threads of execution that use the memory access.

70. The system as recited in claim 69, wherein the race condition detection module includes a user interface configured to prompt for the variable.

71. The system as recited in claim 69, wherein the assertion determination module and

the deassertion determination module each include a depth counter configured to stop execution at a select depth.

72. The system as recited in claim 69, wherein each module is recursively callable.

73. The system as recited in claim 69, wherein multiple instances of each module execute.